View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

Phantm: PHP Analyzer for Type Mismatch

Etienne Kneuss Philippe Suter Viktor Kuncak
School of Computer and Communication Sciences, Swiss Federal Institute of Technology (EPFL)
firstname.lastname@epfl.ch

ABSTRACT

We present PHANTM, a static analyzer that uses a flow-sensitive analysis to detect type errors in PHP applications. PHANTM can infer types for nested arrays, and can leverage runtime information and procedure summaries for more precise results. PHANTM found over 200 true problems when applied to three applications with over 50'000 lines of code, including the popular DokuWiki code base.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; F.3.1 [Logics and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs

General Terms: Languages, Reliability, Verification

1. INTRODUCTION

PHP is a very popular scripting language. PHP scripts are behind many web sites, including wikis, content management systems, and social networking web sites. It is notably used by major web actors, such as Wikipedia, Facebook or Yahoo. Unfortunately, it is easy to write PHP scripts that contain errors. Among the PHP features that are contributing to this fact is the lack of any static system for detecting type or initialization errors.

This paper presents PHANTM,[1] a static analyzer for PHP 5 based on data-flow analysis. PHANTM is an open-source tool written in Scala and available from http://lara.epfl.ch/dokuwiki/phantm. It contains a robust parser that passes 10'000 tests from the PHP test suite and a static analysis algorithm for type errors. PHANTM uses an abstract interpretation domain that approximates values of variables for both simple and structured types, such as arrays and objects. PHANTM is flow-sensitive, which is natural given that the same PHP variable can have different types at different program points.

PHANTM supports a large number of PHP constructs in their most common usage scenarios, with the goal of maximizing the usefulness of the tool. It incorporates precision-enhancing support for several PHP idioms that we frequently encountered and for which our initial approach was not sufficiently precise.

PHANTM analyzes each function separately by default, but uses PHP documentation features to allow users to declare types of function arguments. It also comes with detailed type prototype information for a large number of library functions. We have found PHANTM very helpful in annotating existing code bases. PHANTM also includes a simple version of inter-procedural analysis. Through additional flexibility that goes beyond simple types, PHANTM supports many current programming styles while preventing questionable practices. We therefore hope that it can influence the future evolution of the language, leading to more reliable applications.

[1]PHp ANalyzer for Type Mismatch

Copyright is held by the author/owner(s).
FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.
ACM 978-1-60558-791-2/10/11.

2. RESULTS

PHANTM has been applied to two substantial PHP applications and one library, with a total of over 50'000 lines of PHP code, including a webmail client used by several thousand users, the popular DokuWiki software[2] and the SimplePie news aggregator library.[3] PHANTM identified over 200 problems in the code and its documentation. Some of these problems can cause exploits, infinite loops, and crashes. The total analysis time was 276 seconds, as the following table shows.

	Lines	Warnings	Problems	Time
DokuWiki	31486	270	76	244s
WebMail	3621	59	43	11s
SimplePie	15003	327	84	21s
Total	*50110*	*656*	*203*	*276*s

The Problems column includes bugs, dangerous implicit conversions, statements that issue notices in PHP, and errors in annotations present in the original code. All problems were confirmed by manual examination of the warnings. Note that the analysis time depends more on the code structure than on its size expressed in terms of lines of code.

3. EXAMPLE

We illustrate some of the challenges in type analysis of PHP and show how PHANTM tackles them. Consider the following code, inspired by code bases we have encountered:

```
$conf["readmode"] = "r";
$conf["file"] = fopen($inputFile, $conf["readmode"]);
$content = fread($conf["file"]);
fclose($conf["file"]);
```

Note that several values of different type are stored in an array. To check that the call to the library function **fopen** is correctly typed, we need to establish that the value stored

[2]http://www.dokuwiki.org
[3]http://simplepie.org/

in $conf["readmode"] is a string. Our analysis thus cannot simply abstract the value of $conf as "any array", as the mapping between the keys and the types of the value needs to be stored. On this code, PHANTM correctly concludes that the entry for the key "readmode" always points to a string.

The function **fopen** tries to open a file in a desired mode and returns a pointer to the file (called *resource* in PHP) if it succeeded, and the value **false** otherwise. To handle such code, PHANTM computes the result type of the call as "any resource *or* **false**". Because **fread** expects a resource only, PHANTM displays the following warning:

```
Type mismatch. Expected: Array[file => Resource, ...],
found: Array[file => Resource or False, ...]
```

This warning points to the fact that the code does not properly handle the case when the file cannot be opened. Although **fclose** likewise expects only a resource, PHANTM does *not* emit a second warning for the fourth line. This is because, when it detects a type mismatch, PHANTM applies type refinement on the problematic variable, assuming that the intended type does not go beyond the one expected type. This often eliminates or greatly reduces the number of warnings for the same variable.

Having realized the error thanks to PHANTM's output, we can now change the code to properly handle failures to open the file, as follows:

```
$conf["readmode"] = "r";
$conf["file"] = fopen($inputFile, $conf["readmode"]);
if($conf["file"]) {
  $content = fread($conf["file"]);
  fclose($conf["file"]); }
```

Now that the calls to **fread** and **fclose** are guarded by a check on $conf["file"], PHANTM determines that their argument will never evaluate to **false** and accepts the program as type correct. PHANTM thus takes into account the order of statements and the meaning of conditions.

4. FEATURES

Data-flow analysis lattice. PHANTM abstracts most scalar types by a single value, with booleans being the exception. String and integer constants are abstracted by their precise value when they serve as keys in a map. For example, to denote all maps where the key `"x"` is mapped to an integer and all other keys are undefined PHANTM uses the abstract value $\mathsf{Map}^\sharp[\text{"x"} \mapsto \mathsf{Int}^\sharp, ? \mapsto \mathsf{Undef}^\sharp]$. PHANTM uses allocation-site abstraction to model dynamically allocated objects.

Using null as a value often has a meaning, while reading from unassigned variables is generally an error. To distinguish between these two scenarios, PHANTM uses two different abstract values for these two uses and handles them differently in the transfer function. PHANTM thus incorporates a limited amount of history-sensitive semantics.

PHANTM approximates the set of types that a variable can have at a given program point. To do so, it considers as the abstract domain not only the values representing a specific type (such as $\mathsf{Int}^\sharp$), but also *union types*. For structured union types PHANTM applies a form of independent attribute approximation, replacing a union of map types with a map whose entries have union types.

Built-in Support for Important APIs and Documentation. PHP comes with a large library of functions and classes. The main extensions shipped with PHP contain more than 2'500 functions and classes. PHANTM can correctly represent this internal API, which is a key factor to obtain useful analysis results. This API is stored in an external XML file, allowing easy modifications. Based on multiple call-sites of the user-defined functions, PHANTM can also generate a corresponding API in XML format. This file can then easily be refined by hand, and imported for subsequent analyses to obtain more precise results.

Interprocedural analysis. Instead of defining multiple functions with different names based on the types they accept, PHP functions usually allow many types as input and then dispatch based on them. This causes problems for the source annotations, as they do not allow multiple disjoint prototypes. To solve this issue it is possible to use PHANTM's XML API which allows multiple prototypes. A more automated alternative is to request an interprocedural analysis that reanalyzes functions for each type context. PHANTM can automatically perform context-sensitive analysis of selected functions.

Runtime instrumentation. PHANTM includes a PHP library to instrument the analyzed application. When using this library, the developer uses a function call to indicate a *milestone* in the code, then runs the application using the standard PHP interpreter. Upon reaching the milestone, the library saves in a file a snapshot of all included source files, as well as the values of global and local values at the milestone. It then terminates the execution. The developer can then run PHANTM using this saved state information as an alternative starting point for static analysis, which often produces a higher-quality output. More details and evaluation results related to this feature, as well as more information on PHANTM in general, can be found in [2].

5. RELATED

Existing work on static analysis of PHP primarily focused on specific security vulnerabilities. PIXY [1] is a static analysis tool checking for security vulnerabilities such as cross site scripting (XSS) or SQL injections, which remain the main attack vectors of PHP applications. Wassermann and Su [3] present work on statically detecting SQL injections.

It is only recently that some work considered static analysis of *types* in PHP applications. Notably, the Facebook HIPHOP project[4] is relying on a certain amount of type analysis to optimize the PHP runtime. The recently released tool PHPLINT[5] aims to detect bugs through type errors. Even if its goal is close to the present work, our tool has a much more precise abstract domain, and therefore reports many fewer spurious warnings.

6. REFERENCES

[1] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities. In *IEEE Symp. Security and Privacy*, 2006.

[2] Etienne Kneuss, Philippe Suter, and Viktor Kuncak. Runtime instrumentation for precise flow-sensitive type analysis. In *1st International Conference on Runtime Verification (RV'10)*, 2010.

[3] Gary Wassermann and Zhendong Su. Sound and precise analysis of web applications for injection vulnerabilities. In *PLDI*, 2007.

[4] http://github.com/facebook/hiphop-php/

[5] http://www.icosaedro.it/phplint/